REMARKS

This paper is responsive to the Office Action dated February 24, 2004. All rejections and objections of the Examiner are respectfully traversed. Reconsideration is respectfully requested.

Applicants wish to thank Examiner Opie for his time and helpful consultation in a telephone interview on Feb. 17, 2005. The contents of this paper are intended to reflect the substance of that discussion.

At paragraphs 4-5 of the Office Action, the Examiner rejected claims 1-33 as being obvious under 35 U.S.C. 103, citing the combination of United States patent number 5,509,123 of Dobbins et al. ("Dobbins et al."), United States patent number 5,983,274 of Hyder et al. ("Hyder et al."), and United States patent number 6,810,409 of Fry et al. ("Fry et al."). Applicants respectfully traverse this rejection.

Dobbins et al. disclose an object-oriented architecture which distributes function and system behavior into autonomous router objects. Each router object of Dobbins et al. has three types of imbedded functionality automatically built in, including:

1. The common protocol-independent functions of the object, which may be a routing function (e.g., forwarding information base, FIB) or a system function (e.g., event or timer). For example, all forwarding engines have a forward method as well as a service method regardless of the particular protocol.

2. The functions provided by a base resource object class which define the methods and data for configuration and control. For example, any object of the network interface class has network interface-configuration inside it automatically.

3. The functions provided by the managed object class which define the methods and data for network management.

FIG. 3A of Dobbins et al. is a three-dimensional view of the system architecture according to Dobbins et al. showing the logical relationships between objects, in which each functional subsystem is shown as a separate "plane." As shown in FIG. 3A, the Dobbins et al. system architecture includes four horizontally disposed planes for routing applications, host communications, forwarding, and network interfaces. The Dobbins et al. system utilizes distributed autonomous forwarding engines as shown in FIG. 4. Rather than having a single centralized forwarding engine, in the distributed model of Dobbins et al., each interface has a forwarding engine sitting above it, and each forwarding engine knows how to receive and transmit packets on its own interface. Also, each engine only knows its own configuration information and only knows how to receive and transmit packets on the one interface it is associated with. Each forwarding engine of Dobbins et al. accesses a common forwarding table. Dobbins et al. teaches that the specific goal of the each forwarding engine is to provide the reception, processing, and forwarding of network layer packets, and that at a very high level, all forwarding engines perform these same basic tasks regardless of protocol or media.

The service function of Dobbins et al. refers to the actual in-bound processing of a network layer packet, consisting of reception of the packet from the data link layer, validation and error processing, filter processing, and route lookup processing to determine the next hop. The forward function of Dobbins et al. refers to the actual out-bound processing of a network layer packet, which consists of filter processing, converting network layer address to physical address, and passing the packet to the data link layer of the outbound interface for transmission.

The protocol-specific Forwarding and Service (FAS) engines of Dobbins et al. are derived from a common base class, each protocol forwarding engine has a generic interface for packet servicing and forwarding, regardless of the specific protocol type. Dobbins et al. teaches that this is done on the service side by having each protocol FAS register a pointer to its base class with a packet dispatcher at the data link level for each interface it is instantiated on, thus allowing a packet dispatcher to invoke the overloaded virtual service method without having to know protocol FAS specifics.

To provide an object-oriented, powerful and very efficient access control mechanism, a base class FAC (Forwarding Access) is provided in the Dobbins et al. system, that keeps access list entries as nodes in an AVL tree.

The MIB (Management Information-Base) of Dobbins et al. is structured as a tree of nodes, with each node representing an identifier in the Object Identifier (OID), and Managed Objects placed at the leaf nodes.

Hyder et al. disclose a system for creation and use of control information associated with packetized network data by protocol drivers and device drivers. The Hyder et al. system is designed to allow a software component that processes network data to communicate control information to, and cooperate with, another software component by associating control information with a packet of network data. The Hyder et al. system associates control information with network data upon which control information will operate by appending one or more control data structures to a packet descriptor that is common to all software components processing the network data. The control data structure in Hyder et al. is "tagged" with a class ID value that allows other software components to recognize and utilize the control information. Hyder et al. intend their system to enable any software component to cooperate with and communicate to another software component that processes the network data regardless of any intervening software components.

Hyder et al. use the term "direct call linkage" to refer to a function call interface in their system. As discussed by Hyder et al., address resolution may be done at compile time through traditional linker programs, or may be done dynamically by system components when using such entities as dynamic link libraries or export libraries.

Hyder et al. use the term "direct call linkage" to refer to a function call interface, and address resolution in Hyder et al. may be done at compile time through traditional linkers, or may be done dynamically by system components when using such entities as dynamic link libraries or export libraries. An invocation session in Hyder et al. is created when a subroutine is initially called and ends when that particular subroutine ends. An Application Programming Interface (API) in Hyder et al. is a set of subroutines provided by one software component so that relevant services may be uniformly accessed.

Hyder et al. further teaches that network data may contiguously reside in a memory buffer, or may be accessed by indirect means or structures such as Memory Descriptor Lists (MDL's) that map segments of virtual memory and their physical page size. In either case, all memory is requested in Hyder et al. by the transport protocol driver or the network device driver through a interface call to the integrating component, which in turn manages allocating memory, while the network data is accessible from the packet descriptor. Also included in the packet data structure of Hyder et al. is a pointer to a series of control data structures, each of which is created by an originating software component, such as the transport protocol driver, the network card device driver, or other component that in some way processes the packet. Each control data structure of Hyder et al. is filled with control information that will eventually be used by another software component other than the component creating the data structure. The packet descriptor of Hyder et al. functions as an integrating data structure to tie together the network data and the series of control data structures.

Fry et al. teaches that the main reason for using Java when building components of the Fry et al. system is that Java programs can be compiled into platform independent byte code which is then interpreted on a Java Virtual Machine, and that Java Virtual Machine (VMs) exist for most commonly used computer hardware architectures.

Nowhere in the combination of Dobbins et al., Hyder et al., and Fry et al., taken either individually or in combination, is there disclosed or suggested any application programming interface to a forwarding plane for processing data packets in a network device that forwards packets across a network, the application programming interface, including:

> an input module that receives routing platform independent function calls, wherein the function calls include routing platform independent input control data, ***and wherein the routing platform independent input control data includes table data to update a routing table***;
>
> at least one control module that receives the input control data via the function calls, the at least one control module producing routing platform specific output control data based upon the input control data, the output control data being capable of controlling execution of the forwarding plane, ***and wherein the at least one control module comprises a plurality of objects arranged in a hierarchical tree structure, the function calls instantiating at least one of the objects for storing the output control data in a memory device***; and
>
> an output module that forwards the output control data from the at least one control module. (emphasis added)

as in the present independent claim 1. Analogous features are also found in the present independent claims 9, 18, 21 and 29. Applicants submit that the present claims are not directed merely to a generic Application Programming Interface (API) as is generically described by

Hyder et al. as "a set of subroutines provided by one software component so that relevant services may be uniformly accessed". Neither is the presently claimed invention the same as a Java VM, which is described by Fry et al. as a reason for using the Java programming language.

The combination of Dobbins et al., Hyder et al., and Fry et al. fails to provide any hint or suggestion of even the desirability of providing an input module that receives routing platform independent function calls, wherein the function calls include routing platform independent input control data, wherein the routing platform independent input control data includes table data to update a routing table, in combination with at least one control module that receives the input control data via the function calls, the at least one control module producing routing platform specific output control data based upon the input control data, the output control data being capable of controlling execution of the forwarding plane, and wherein the at least one control module comprises a plurality of objects arranged in a hierarchical tree structure, the function calls instantiating at least one of the objects for storing the output control data in a memory device, as in the present independent claims. In contrast, Dobbins et al. describes the use of a Routing Protocol to update forwarding information in an FIB, and shows in Fig. 3 a forwarding table 233 accessed within a forwarding plane. The forwarding engines that share the forwarding table 233 are described as each having its own interface. This is distinct from the present independent claims, which provide an application programming interface that is shared across multiple router applications because it accepts *routing platform independent control inputs.* The interfaces to the forwarding table of Dobbins et al. are forwarding engine specific. Moreover, the MIB description in Dobbins et al. includes nothing that along these lines as well.

For the reasons stated above, Applicants respectfully urge that the combination Dobbins et al., Hyder et al., and Fry et al. does not disclose or suggest all the features of the present independent claims 1, 9, 18, 21 and 29. Accordingly, Applicants respectfully urge that the combined references do not support a *prima facie* case of obviousness under 35 U.S.C. 103 with regard to in claims 1, 9, 18, 21 and 29. As to the remaining claims, they each depend from claims 1, 9, 18, 21 and 29, and are respectfully believed to be patentable over the combined references for at least the same reasons. Reconsideration of all pending claims is respectfully requested.

As all claims are believed to be allowable, the application is believed to be in condition for allowance. Withdrawal of the claim rejections and favorable action are respectfully requested.

Applicants have made a diligent effort to place the claims in condition for allowance. However, should there remain unresolved issues that require adverse action, it is respectfully requested that the Examiner telephone the undersigned Attorney at 617-630-1131 so that such issues may be resolved as expeditiously as possible.

For these reasons, and in view of the above amendments, this application is now considered to be in condition for allowance and such action is earnestly solicited.

Respectfully Submitted,

FEB. 22 2005
Date

[illegible]
David A. Dagg, Reg. No. 37,809
Attorney/Agent for Applicant(s)
Steubing McGuinness & Manaras LLP
125 Nagog Park Drive
Acton, MA 01720
(978) 264-6664

Docket No. 2204/A34 120-209